```
Require Import List.
Require Import String.
Require Import ZArith.

Open Scope list_scope.
Open Scope string_scope.
Open Scope Z_scope.

Require Import ImpSyntax.
Require Import ImpCommon.
Require Import ImpEval.

Inductive step (env : env) :
  store -> heap -> stmt ->
  store -> heap -> stmt -> Prop :=
| step_set :
    forall s h x e v,
      eval_e s h e v ->
      step env
        s h (Sset x e)
        (update s x v) h Snop
| step_alloc :
    forall s h x e1 e2 i v,
      eval_e s h e1 (Vint i) ->
      eval_e s h e2 v ->
      0 <= i ->
      step env
        s h (Salloc x e1 e2)
        (update s x (Vaddr (zlen h))) (alloc h i v) Snop
| step_write :
    forall s h x e1 e2 a l i v h',
      lkup s x = Some (Vaddr a) ->
      read h a = Some (Vint l) ->
      eval_e s h e1 (Vint i) ->
      eval_e s h e2 v ->
      0 <= i < l ->
      write h (Zsucc (a + i)) v = Some h' ->
      step env
        s h (Swrite x e1 e2)
        s h' Snop
| step_call_internal :
    forall s h x f es params body ret vs s',
      locate env f = Some (Func f params body ret) ->
      evals_e s h es vs ->
      updates store_0 params vs = Some s' ->
      step env
        s h (Scall x f es)
        s' h (Sincall body ret x s)
| step_call_external :
    forall s h x f es vs h' v',
      locate env f = None ->
      evals_e s h es vs ->
      extcall_spec f vs h v' h' ->
      step env
        s h (Scall x f es)
        (update s x v') h' Snop
| step_ifelse_t :
    forall s h e p1 p2,
      eval_e s h e (Vbool true) ->
      step env
        s h (Sifelse e p1 p2)
        s h p1
| step_ifelse_f :
    forall s h e p1 p2,
      eval_e s h e (Vbool false) ->
      step env
        s h (Sifelse e p1 p2)
        s h p2
| step_while_t :
```



```
    forall s h e p,
      eval_e s h e (Vbool true) ->
      step env
        s h (Swhile e p)
        s h (Sseq p (Swhile e p))
| step_while_f :
    forall s h e p,
      eval_e s h e (Vbool false) ->
      step env
        s h (Swhile e p)
        s h Snop
| step_seq_nop :
    forall s h p2,
      step env
        s h (Sseq Snop p2)
        s h p2
| step_seq :
    forall s h p1 p2 s' h' p1',
      step env
        s h p1
        s' h' p1' ->
      step env
        s h (Sseq p1 p2)
        s' h' (Sseq p1' p2)
| step_incall_ret :
    forall s h e lr sr v,
      eval_e s h e v ->
      step env
        s h (Sincall Snop e lr sr)
        (update sr lr v) h Snop
| step_incall :
    forall s h p e lr sr s' h' p',
      step env
        s h p
        s' h' p' ->
      step env
        s h (Sincall p e lr sr)
        s' h' (Sincall p' e lr sr).

Inductive step_star (env : env) :
  store -> heap -> stmt ->
  store -> heap -> stmt -> Prop :=
| step_star_refl :
    forall s h p,
      step_star env
        s h p
        s h p
| step_star_l :
    forall s1 h1 p1 s2 h2 p2 s3 h3 p3,
      step env
        s1 h1 p1
        s2 h2 p2 ->
      step_star env
        s2 h2 p2
        s3 h3 p3 ->
      step_star env
        s1 h1 p1
        s3 h3 p3.

Inductive steps_p : prog -> val -> Prop :=
| steps_prog :
    forall funcs main ret s' h' v,
      step_star funcs
        store_0 heap_0 main
        s' h' Snop ->
      eval_e s' h' ret v ->
      steps_p (Prog funcs main ret) v.
```